Inductive learning of knowledge-based planning operators

Summary

This thesis addresses the research question:

"Where do planning operators come from?"

Planning operators are data-structures that represent some knowledge about classes of possible actions in a problem domain. They are used in *plan generation*, which is defined as the process of selecting and instantiating actions from a set of planning operators and logically ordering these instantiated actions in a sequence that will, on execution, transform a given initial domain state into a desired ("goal") state.

Until now, planning system developers have been responsible for providing suitable planning operators for a given problem domain, in the same way as knowledge engineers have been responsible for providing suitable production-rules in expert systems. Before domain-specific knowledge can be provided, it must be acquired. In the expert-systems field it has been known since the early 1980s that *knowledge acquisition* is a difficult and laborious process. There has been progress in automating the formulation of production-rules, especially by inducing production-rules from examples. In the non-AI discipline of *software engineering*, the laborious and error-prone nature of the analogous process of *requirements analysis* has been recognised for at least three decades. Recently, similar difficulties have been acknowledged to exist in the compilation of planning knowledge (Minton and Zweben, 1993). The analogy with the induction of production-rules suggests the possibility of inducing planning operators from examples. This possibility motivated my research.

In this thesis, I claim that planning operators can be induced *ab initio* from a domain model represented as collections of domain objects, inter-object relationships, and interrelationship constraints. Moreover, such a domain model can be extracted from a collection of state-descriptions.

I substantiate this claim by developing an induction algorithm - known as the *Planning Operator Induction* (POI) algorithm - by implementing the algorithm, and by performing various experiments on the implemented programs. To do this, I had to develop:

- *the POI ontology.* An *ontology* is a set of definitions of content-specific knowledge-representation primitives that is both human and machine readable (Gruber, 1992). The POI ontology is based on primitives found in the *entity-relationship model* (Chen, 1976) of domain structure and in the *state-transition model* of domain change (Shlaer and Mellor, 1992).

- *a family of meta-heuristics.* A *meta-heuristic* is a form of control structure that applies to a set of domains (Sowa, 1984), i.e., it is domain-independent.

My research differs from other approaches to learning planning knowledge in that it:

- *learns domain knowledge.* Most of the research to date has concentrated on learning control knowledge, i.e., knowledge indicating how the planning system should control its search for a plan during plan generation.

- *induces behavioural knowledge from structural knowledge.* Other approaches have been restricted to behavioural knowledge alone.

- *represents the induced knowledge as planning operators.* A variety of knowledge representations are used in other induction systems, e.g., production rules.

- *uses an ontology grounded on mature software engineering principles.* Other researchers have largely failed to document their ontologies.

- *enables the induction of behavioural domain knowledge ab initio.* Most research has been concerned with the easier task of automating knowledge refinement (Carbonell and Gil, 1990).

- *does not make use of any control knowledge.* Other algorithms depend, at least in part, on inputs that provide domain-specific information on the sequencing of states and/or actions.

There are six chapters in this thesis. Additional material includes a references section, an index, a summary (in English and in Dutch), and my curriculum vitae.

Chapter 1 opens with the motivation for my research, states my thesis claim, and places my research in context. Knowledge-based planning, software engineering, machine learning, and multi-agent systems techniques are identified as related fields. The POI algorithm is outlined and distinguished from plan generation. Potential applications of POI are as a knowledge acquisition tool for planners, as an element of multi-agent systems, as a Computer Aided Software Engineering tool, as a commercial or military intelligence support tool, and as an instruction support tool for teachers. The POI algorithm is limited in that it does not guarantee perfection, efficiency, optimality, or applicability to all problems and domains. Aspects of the related fields which are specifically excluded from my research are:

- scheduling, design, or other planning specialisations involving metric quantities;

- uncertainty arising from forgetful agents, from imprecise, incorrect, or conflicting domain knowledge, or from incompleteness in initial or goal state descriptions;

- the selection of series of observations;

- the recognition of objects and their classes in world-state observations.

The extensive use of software engineering ideas and methods is emphasised in taking an engineering approach to my research. The history of my research is summarised. Finally, the thesis conventions and layout are stated.

Chapter 2 reviews the relevant literature in the related fields. Software engineering and knowledge-based planning were the sources of the POI ontology. The types of knowledge that need to be represented in the behavioural part of the ontology were identified from Tate, Hendler and Drummond's (1990) definition of plan generation. The structural part was based on Chen's (1976) entity-relationship model, enhanced by means of Nijssen and Halpin's (1989) exclusion constraints. The aspects of the POI ontology that are unusual from the knowledge-based planning viewpoint are:

- the provision of a structural domain model;

- the description of states in terms of inter-object relations;

- the representation of plans as sequences of states, rather than sequences of transitions (i.e.,

instantiated operators); and

- the emphasis on representing domain constraints.

Machine learning is the source of the POI's core inferencing process. The POI algorithm encapsulates an inductive technique known as the *version space and candidate elimination* algorithm (Mitchell, 1982). Other learning systems in the planning area are surveyed, showing that they either learn control knowledge or make use of sequencing information. Multi-agent systems techniques were used in *closed-loop* testing of the POI algorithm. Single- and multi-agent learning are contrasted, and the role of inter-agent cooperation in learning is summarised.

Chapter 3 documents the POI algorithm and its underlying ontology. Functionality, application, and tractability requirements are compiled. The POI ontology connects a structural model of the problem domain to its behavioural model by means of a linking model. The ontology is presented using Chen's (1976) entity-relationship notation. Since the POI algorithm also takes its input in the form of an entity-relationship model of the domain, the POI ontology is a *meta-representation.*

Following accepted software engineering practice, the POI algorithm is described in terms of its top-down, functional decomposition. At the top level, the algorithm is decomposed into two Parts. At the second level of decomposition, the algorithm consists of nine steps. Each step is described in detail, with some steps being decomposed to a third level. Single- and multi-agent implementations of the POI algorithm are described, with the implementation-language classes being related to the entity-classes in the POI ontology. The behaviour of the single-agent program is shown as a state-transition network. Key refinements made during implementation are outlined.

The limitations of the POI algorithm are described. The most serious limitation is that the algorithm is inherently combinatorially explosive. The algorithm's complexity is analysed theoretically, showing that the worst-case runtime and memory usage would be proportional to 2 to the power of the number of sentences in the state-description language. Potential countermeasures to the combinatorial explosion are described, including the use of domain-specific heuristics, a plan-space lattice, a class-level version space, inheritance, whole-part decomposition, and version-space partitioning. The inheritance and version-space partitioning countermeasures have been implemented.

Chapter 4 describes a series of *open-loop* experiments designed to:

- demonstrate that the POI algorithm was capable of inducing planning operators;

- investigate the algorithm's complexity empirically.

Both programme goals have been achieved. Using the single-agent program, sets of planning operators have been induced for eight domains ranging from one to 24 object-classes. Extracts of the outputs for three selected domains have been documented. Planning-operator sets to be found in the AI literature can be reproduced. Furthermore, novel planning operators can be induced, as shown using the High Performance Capillary Electrophoresis domain. Several of these novel operators could not have been learned by other operator-learning algorithms.

The sensitivity of the POI algorithm has been investigated by varying the numbers of object-classes, object-instances, relationship-classes, and domain constraints, and by applying each of the four meta-heuristics in turn. The results of these investigations are summarised. The complexity of the POI algorithm is investigated empirically, both within a given domain and across several domains. Within-domain complexity was found to be consistent with exponential behaviour, and across-domain

complexity is consistent with a third-order polynomial.

The effectiveness of the countermeasures was compared with the baseline algorithm. For small numbers of object-classes and of object-instances per class, the introduction of inheritance was found to be more effective than version-space partitioning. Combining the countermeasures is more effective than either countermeasure alone. The introduction of inheritance has also been shown to be effective in enabling domain constraints to be modelled completely using binary exclusion-rules. Finally, the fact that the POI ontology is a meta-representation was exploited to perform *meta-runs*, i.e., POI runs for which POI itself was the problem (meta-)domain.

Chapter 5 describes a second series of experiments designed to close the loop in testing the POI algorithm. The loop was closed by embedding the full POI algorithm, together with reactive and deliberative planning functionalities, in a *POIAgent*. One or more POIAgents were placed in an environment consisting of several other simpler agents, which simulated the POIAgents' problem domain. At the start of each run, the POIAgents were *naive*, i.e., they had no prior domain knowledge. Each POIAgent was given a series of goals to achieve by interacting with the simpler agents in its environment. The goal-series were designed so that the POIAgents acquired knowledge about their environment, induced a set of planning operators from the acquired knowledge, generated a plan using the induced operators, and executed that plan successfully. In short, the POIAgents performed *learning-by-doing* (Anzai and Simon, 1979).

Some experiments were done with a single POIAgent, to investigate single-agent learning in a multi-agent environment. Using the blocks world (Winograd, 1972), I showed that, given a selected pair of 3-blocks-world states, a POIAgent induced correctly the complete set of planning operators from Nilsson (1980). Other experiments were done with two POIAgents, to investigate multi-agent learning. Neither POIAgent gained complete knowledge of the overall domain by its own observations. They had to exchange information they had acquired about their individual problem domains. The information exchanged was represented as a domain model. To perform *learning-by-being-told*, recipient POIAgents had to be given additional functionality to *assimilate* (Lefkowitz and Lesser, 1990) the received domain models with their own domain model.

Chapter 6 summarises the results of my research. It concludes that it is feasible to:

- automate the learning of planning operators.
- induce planning operators *ab initio* from collections of domain objects, inter-object relationships, and interrelationship constraints, as claimed.
- compile such lists from unordered observations of non-adjacent domain states.
- induce planning operators for complex, real-world domains.

The contributions of my research are summarised for each of the related fields. Directions for future research are indicated. Finally, the significance of my research, both in AI and in software engineering, is identified. For AI, the research makes it possible to generate automatically sets of planning operators consistent with the structural model of a domain. For software engineering, the research makes it possible to generate a state-transition network from an entity-relationship model. This could both save effort in software design and reduce errors arising from inconsistency between the structural and behavioural models.

Samenvatting

Titel in het Nederlands:

Inductief leren van op kennis gebaseerde planningsoperatoren

Dit proefschrift beschrijft onderzoek naar de vraag:

"Waar komen planningsoperatoren vandaan?"

Planningsoperatoren zijn datastructuren die kennis representeren over klassen van mogelijke acties in een toepassingsdomein. Ze worden gebruikt in het planningsproces. *Op kennis gebaseerde planning* is gedefineerd als het proces van selecteren en instantiëren van acties vanuit een verzameling planningsoperatoren, om vervolgens de geïnstantieerde acties in een logische volgorde te zetten. Na uitvoering van de acties is een initiële domeintoestand omgezet in een gewenste ("doel") toestand.

Tot nu toe zijn de ontwikkelaars van planningssystemen verantwoordelijk geweest voor het ontwerp van de juiste planningsoperatoren voor een bepaald toepassingsdomein, en wel op dezelfde manier als kennistechnologen verantwoordelijk zijn voor relevante regels in expertsystemen. Voordat relevante domein-specifieke kennis geleverd kan geworden, moet zij eerst verworven worden. Binnen het gebied van expertsystemen weten we sinds het begin van de tachtiger jaren dat het verwerven van kennis een moeilijk en bewerkelijk proces is. Wel is er vooruitgang geboekt met het automatiseren van het formuleren van productieregels, met name in de inductie van productieregels aan de hand van voorbeelden. Buiten de AI staat een vergelijkbaar proces van analyse van gebruikerseisen bekend als bewerkelijk en foutgevoelig. In de laatste jaren is duidelijk geworden dat binnen de AI overeenkomstige moeilijkheden bestaan bij het verwerven van planningskennis (Minton en Zweben, 1993). De analogie van planningssystemen met expertsystemen suggereert dat het goed mogelijk is om planningsoperatoren vanuit voorbeelden te induceren. Deze mogelijkheid was de motivatie voor mijn onderzoek.

In dit proefschrift beweer ik dat het mogelijk is om planningsoperatoren te induceren vanaf het begin vanuit een domeinstructuurmodel dat wordt gerepresenteerd door verzamelingen van domein-objecten, relaties tussen objecten, en constraints tussen relaties. Zulk een domeinstructuurmodel kan worden verkregen vanuit een verzameling toestandsbeschrijvingen.

Om mijn bewering te onderbouwen heb ik een inductie-algoritme ontwikkeld: het *Planning Operator Induction* (POI) algoritme. Het POI-algoritme is geïmplementeerd en er zijn verscheidene experimenten mee uitgevoerd. Voor de ontwikkeling van het algoritme heb ik tevens ontworpen:

- *De POI-ontologie.* Een *ontologie* is een verzameling definities van kennisrepresentatie-primitieven die zowel voor mensen als voor machines leesbaar is (Gruber, 1992). De POI-ontologie is voor de domeinstructuur gebaseerd op primitieven van Chen's (1976) *entiteit-relatie model* en voor het domeingedrag op primitieven van het *toestandsmodel* (Shlaer en Mellor, 1992).

- *Een familie van meta-heuristieken.* Een *meta-heuristiek* is een besturingsstructuur die toepasbaar is op een verzameling domeinen (Sowa, 1984), dat wil zeggen de meta-heuristiek is domeinonafhanklijk.

Mijn onderzoek wordt gekenmerkt door de volgende verschillen met andere aanpakken voor het leren

van planningskennis, doordat het:

- *domeinkennis leert.* De meerderheid van onderzoek tot nu toe heeft zich gericht op het leren van besturingskennis, dat wil zeggen kennis die tijdens het planningsproces het zoekproces bestuurt.

- *gedragskennis van structuurkennis induceert.* Ander onderzoek is beperkt tot alleen gedragskennis.

- *de geïnduceerde kennis als planningsoperatoren representeert.* Er zijn verschillende representaties in gebruik in andere inductieve systemen, bijvoorbeeld productie-regels.

- *een ontologie gebruikt die gebaseerd is op voldragen software-engineering principes.* In het algemeen hebben andere onderzoekers zijn ontologieën niet gedocumenteerd.

- *induceren van gedragskennis mogelijk maakt vanaf het begin.* Het meeste onderzoek betreft de minder moeilijke taak van het automatiseren van kennisverfijning (Carbonell en Gil, 1990).

- *geen gebruik van besturingskennis maakt.* Andere algoritmen zijn afhankelijk, gedeeltelijk of in zijn geheel, van invoer-informatie die domein-specifieke kennis geeft over de volgorde van toestanden en/of acties.

Dit proefschrift bestaat uit zes hoofdstukken. Bijgevoegd materiaal bestaat uit referenties, samenvattingen in het Engels en Nederlands, mijn curriculum vitae, en een trefwoordenregister.

Hoofdstuk 1 formuleert de motivatie en probleemstelling van deze studie en plaatst het onderzoek in zijn context. Op kennis gebaseerde planning, software-engineering, lerende systemen en multi-agent systemen worden als gerelateerde gebieden geïdentificeerd. Het POI-algoritme wordt beschreven en de verschillen met planning worden duidelijk gemaakt. Als potentiële toepassingen noemen we een kennisacquisitie-hulpmiddel voor planners, een functioneel element van multi-agent systemen, een Computer Aided Software Engineering hulpmiddel, een ondersteunend hulpmiddel voor commerciële of militaire inlichtingen, en een ondersteunend hulpmiddel voor leraren. Het POI-algoritme is beperkt in die zin dat het geen garantie biedt voor volledigheid, efficiency, optimaliteit, of toepasbaarheid voor alle mogelijke problemen en domeinen. Aspecten van de gerelateerde gebieden die niet voorkomen in mijn onderzoek zijn:

- scheduling, ontwerpen, of andere specialisaties van planning met metrische eenheden;

- onzekerheid die afkomstig is van agenten die kunnen vergeten, van domeinkennis die onnauwkeurig, incorrect of conflicterend is, of van initiële- of doelstandstanden die onvolledig zijn;

- het selecteren van een reeks toestandswaarnemingen;

- het herkennen van objecten en hun klassen in de toestandswaarnemingen.

Mijn aanpak, die sterk gebaseerd is op begrippen en methoden uit de software-engineering, wordt precies uitgelegd. De geschiedenis van mijn onderzoek wordt geschetst. Conventies en structuur van het proefschrift worden als leidraad gegeven.

Hoofdstuk 2 laat de relevante literatuur in de gerelateerde gebieden de revue passeren. Software-engineering en op kennis gebaseerde planning zijn de bronnen voor de POI-ontologie. De typen van domeingedragkennis zijn ontleend aan Tate, Hendler en Drummond's (1990) definitie van planning. De typen van domeinstructuurkennis zijn gebaseerd op Chen's (1976) entiteit-relatie model, uitgebreid met Nijssen en Halpin's (1989) uitsluitingsconstraints. Aspecten van de POI-ontologie die ongewoon zijn vanuit het oogpunt van op kennisgebaseerde planning zijn:

- het leveren van een domeinstructuurmodel;
- de beschrijving van domeintoestanden in termen van relaties tussen domeinobjecten;
- de representatie van plannen als een reeks van toestanden, in plaats van een reeks van acties (dat wil zeggen geïnstantieerde operatoren); en
- de nadruk op het representeren van domeinconstraints.

De *version space and candidate elimination* techniek van lerende systemen (Mitchell, 1982) is de kern van het POI-algoritme. Andere technieken bij lerende systemen in het planningsgebied gebruiken òf besturingskennis òf kennis over de volgorde van de operatoren. Multi-agent systeemtechnieken worden gebruikt in het *closed-loop* testen van het POI-algoritme. Het contrast tussen enkel- en multi-agent leren wordt aangegeven, en de rol van inter-agent samenwerking in het leerproces wordt samengevat.

Hoofdstuk 3 documenteert het POI-algoritme en de POI-ontologie. Functionele, toepassings-, en handelbaarheidseisen worden bijeengebracht. De POI-ontologie verbindt het domeinstructuurmodel met het domeingedragmodel door middel van een verbindingsmodel. Chen's (1976) entiteit-relatie model wordt gebruikt om de POI-ontologie te representeren. Omdat het POI-algoritme een entiteit-relatie model van een domein als invoer neemt is de POI-ontologie een *meta-representatie*.

Het POI-algoritme is ontworpen door middel van een bekende software-engineering techniek, namelijk de top-down, functionele decompositie. Op de hoogste niveau zijn er twee delen en op de tweede niveau zijn er negen stappen. Sommige stappen zijn te ontleden tot op een derde niveau. Elke stap in het algoritme wordt in detail beschreven. Twee implementaties worden gepresenteerd, namelijk de single-agent implementatie en de multi-agent implementatie. De relatie tussen klassen in de POI-ontologie en klassen in de implementatietaal wordt daarbij uitgelegd. Het gedrag van de single-agent implementatie wordt beschreven door middel van een toestandsdiagram. Verfijningen gemaakt tijdens de implementatie worden geschetst.

De beperkingen van het POI-algoritme worden geïdentificeerd. Het belangrijkste hierbij is dat het algoritme combinatorisch-explosief is. De complexiteit van het algoritme wordt theoretisch geanalyseerd, met als resultaat dat, in het ergste geval, tijd en geheugen proportioneel zijn met twee tot de macht van het aantal zinnen in de domeinbeschrijvingstaal. Diverse tegenmaatregelen worden geformuleerd, waaronder het gebruik van domein-specifieke heuristieken, van roosters in planningsruimten, van version space op het niveau van een klasse, van overerving, van decompositie, en van version-space verdeling. Twee tegenmaatregelen zijn geïmplementeerd, namelijk overerving en version-space verdeling.

Hoofdstuk 4 beschrijft een serie van *open-loop* experimenten. Deze experimenten zijn ontworpen om twee doelen te bereiken:

- te demonstreren dat het POI-algoritme planningsoperatoren kan induceren;

- de werkelijke complexiteit te onderzoeken.

Beide doelen worden gehaald. Planningsoperatoren zijn voor acht domeinen geïnduceerd, met een omvang variërend van 1 tot 24 object-klassen. Delen van de uitvoer voor drie geselecteerde domeinen zijn in dit proefschrift gedocumenteerd. Planningsoperatoren uit de AI-literatuur kunnen gereproduceerd worden. Verder kunnen operatoren geïnduceerd worden die nooit eerder zijn beschreven, zoals wordt aangetoond met behulp van het High Performance Capillary Electrophoresis domein. Verschillende van die nooit-eerder-geïnduceerde operatoren kunnen andere operator-lerende algoritmen niet genereren.

Door het aantal object-klassen, object-instanties, relatie-klassen, en domein-constraints te variëren, en door elke meta-heuristiek toe te passen bestuderen we de gevoeligheid van het POI-algoritme. De resultaten zijn samengevat. De werklijke complexiteit, zowel binnen een bepaald domein als tussen domeinen, is gemeten. De complexiteit binnen een domein komt overeen met exponentieel gedrag, en de complexiteit tussen de domeinen komt overeen met een polynoom van de derde orde.

De effectiviteit van de tegenmaatregelen wordt vergeleken met de effectiviteit van het baseline-algoritme. Voor een beperkt aantal object-klassen en object-instanties per klasse is het introduceren van overerving effectiever dan version-space verdeling. Een combinatie van tegenmaatregelen blijkt effectiever dan één maatregel alleen. Overerving maakt het ook mogelijk om alle domein-constraints te representeren door middel van binaire uitsluitingsregels. Tenslotte wordt het feit dat de POI-ontologie een meta-representatie is, uitgebuit door experimenten te doen waarin POI tevens het toepassingsdomein is.

Hoofdstuk 5 beschrijft een serie van *closed-loop* experimenten. Het POI-algoritme wordt daarvoor omgeven door toegevoegde functionaliteiten voor reactieve en generatieve planning in een zogenaamde *POIAgent*. Eén of twee POIAgenten worden geplaatst in een omgeving met meerdere, eenvoudige agenten, die het POIAgentendomein nabootsen. Elke POIAgent is in het begin naïef, dat wil zeggen zij heeft geen kennis over het domein. Iedere POIAgent is voorzien van een serie taken die uitgevoerd moet worden door interactie met de eenvoudige agenten. De taken zijn zo ontworpen dat de POIAgenten kennis over het domein verwerven, operatoren induceren, plannen genereren met die operatoren, en die plannen uitvoeren. In het kort, de POIAgenten presteren volgens *learning-by-doing* (Anzai en Simon, 1979).

Sommige experimenten worden met één POIAgent gedaan. Met behulp van het domein van de blocks world (Winograd, 1972) toon ik aan dat een POIAgent de volledige verzameling planningsoperatoren, zoals door Nilsson (1980) gedefinieerd is, kan induceren. Andere experimenten worden met twee POIAgenten gedaan. Geen van beide POIAgenten kon hierbij afzonderlijk de volledig kennis vergaren, maar elk van hen moest informatie uitwisselen om tot een volledige verzameling planningsoperatoren te komen. De informatie wordt als een domeinstructuurmodel uitgewisseld. Om *learning-by-being-told* te implementeren is het nodig dat aan elke POIAgent een functionaliteit wordt toegevoegd, zodat een POIAgent het domeinstructuurmodel met zijn eigen domeinstructuurmodel kan *assimileren* (Lefkowitz en Lesser, 1990).

Hoofdstuk 6 vat de resultaten van mijn onderzoek samen. De belangrijke conclusies zijn:

- het leren van planningsoperatoren kan geautomatiseerd worden;

- planningsoperatoren kunnen inderdaad, zoals ik beweerd heb, geïnduceerd worden vanaf het begin vanuit verzamelingen domein-objecten, relaties tussen objecten, en constraints tussen relaties;

- soortgelijke domeinstructuurmodellen kunnen verkregen worden vanuit een verzameling toestandsbeschrijvingen;

- planningsoperatoren kunnen geïnduceerd worden voor complexe praktijkdomeinen.

De bijdrage van mijn onderzoek tot ieder afzonderlijk gerelateerd gebied wordt samengevat. Indicaties worden gegeven voor mogelijk toekomstig onderzoek door anderen. Tenslotte wordt de betekenis van mijn onderzoek geïdentificeerd voor zowel de AI als voor de software-engineering. Voor de AI maakt mijn onderzoek het mogelijk om planningsoperatoren consistent met een domeinstructuurmodel automatisch te genereren. Voor de software-engineering maakt mijn onderzoek het mogelijk om een toestandsdiagram automatisch te genereren vanuit een entiteit-relatie model. Dit kan zowel inspanning in het ontwerpen van software besparen, als een bron van menselijke fouten elimineren.